# The Agilent Technologies Instrument Control Framework

An open systems approach to simplify instrument control of Agilent HPLC's in a third-party chromatographic data system or workstation environment.

Technical Overview

## Introduction

This Technical Overview describes the Agilent Instrument Control Framework (ICF), a software component framework, that allows third-party software to enable and control Agilent liquid chromatography (LC) systems with chromatographic data systems or workstations. It outlines the underlying technical concepts of the Rapid Control.NET instrument drivers (RC.NET) and describes how the software components interact in an open systems environment.

In today's competitive environment, many labs must reduce costs, limit training time, and simplify regulatory compliance. The implementation of a single vendor's chromatographic data system throughout the lab is one way to accomplish these goals. If a lab is standardized on a non-Agilent data system, it can still use its favorite Agilent LCs. To achieve this, the software provider can take advantage of Agilent's Instrument Control Framework (ICF), which accelerates and simplifies the development effort for third-party software control of Agilent LCs.

### In summary: what are the advantages of ICF for the user?

If the third-party chromatographic data system or workstation vendor uses ICF to implement control of Agilent LCs, the benefits of ICF become apparent:

- Offers the same look and feel in any third-party LC or LC/MS data system or workstation
- Supports current Agilent 1100 Series, Agilent 1200 Series, Agilent 1220 Infinity, Agilent 1260 Infinity LC modules, and Agilent 1290 Infinity LC modules
- Enables control of new modules or systems at the time they become available from Agilent
- Delivers comprehensive user interfaces that seamlessly incorporate all instrument features, including method editing, status, and configuration
- Provides a combined system view, which gives much better overall status than module-specific views

Agilent Technologies

Instrument Control Framework reflects Agilent's commitment to open systems. Before ICF, third-party software developers used low level instrument control code to write their own native drivers for Agilent LCs. Agilent ICF minimizes this effort. ICF contains instrument drivers and user interfaces (Figure 1), and provides a simple programming interface for third-party software connectivity. For full control of current and future Agilent LCs, the software vendor simply develops an adapter from its own software to **ICF**, with no need to write additional instrument control code. How this can be achieved is described below in more detail.

## Instrument Control – General Overview

A typical analytical application can be characterized with regard to instrument control by the following aspects:

- Overall architecture
- Functional blocks
- Deployment strategies

The architecture of an analytical application is defined by the abstraction layers, which are typically used to implement instrument control for modular or monolithic instruments (Figure 2).

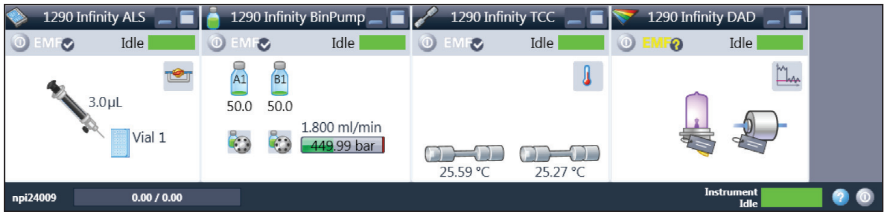
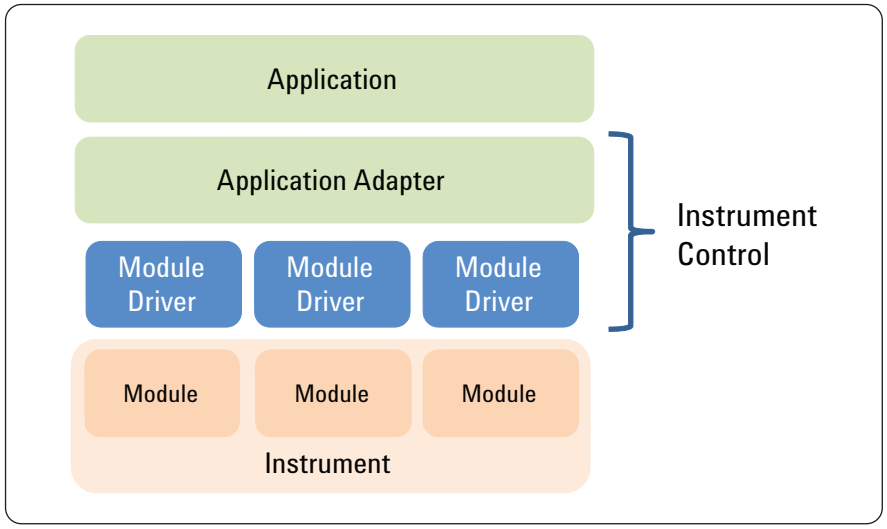



**Figure 1**
**Instrument control user interface for an Agilent 1290 Infinity LC System.**




| | |
|---|---|
| Instrument | Analytical hardware; can be a monolithic instrument with hardware options or a modular instrument, which is built up by combining multiple HW modules. |
| Module Drivers | Software layer, which encapsulates module control specific functionality such as communication protocol handling. |
| Application Adapter | Software adapter, which translates the instrument drivers interface to the application interface. |
| Application | The core analytical application. Typically in charge of automation control, which for instance includes the execution of sequences. |

**Figure 2**
**General overview of instrument control.**

## Functional blocks

From a functional perspective the typical instrument control is comprised of the following functional blocks (Table 1).

## Deployment models

The deployment strategy of an analytic application may vary from a single node (workstation) to a distributed (client/server) deployment (Figure 3a, b).

| | |
|---|---|
| Instrument configuration | Comprises the description and the detection of the physical instrument with regard to options and setup. |
| Instrument method handling | Defines the handling of the set of parameters, which is relevant for an analytical run. |
| Instrument status representation | Reflects the current status of an instrument. |
| Instrument run control | Handles both the physical communication and the control of an analytical run including the acquisition of the measured data from the instrument. |

**Table 1**
**Functional blocks.**

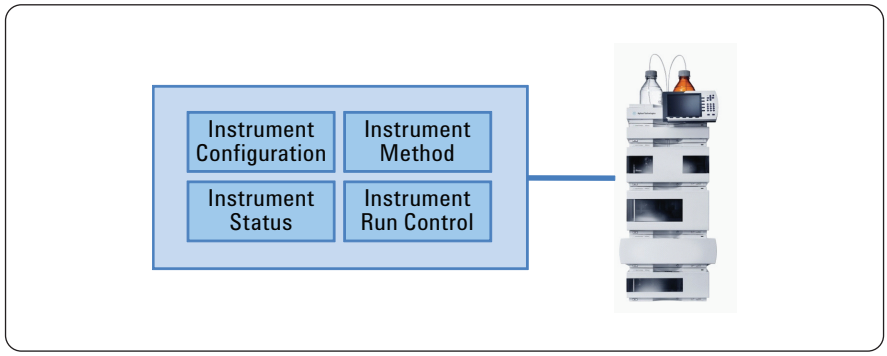



**Figure 3a**
**Example for a typical workstation deployment: All functional blocks reside on a single machine, including the instrument run control.**

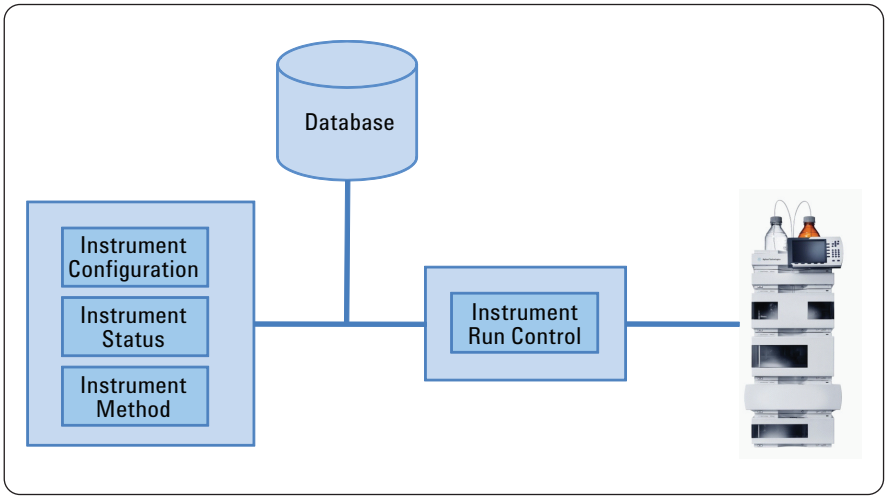



**Figure 3b**
**Example for a client/server deployment:**
**The functional blocks except the instrument run control are distributed within client machines. The instrument run control resides in this example on a distinct machine acting as acquisition controller. The instrument related data such as instrument configuration or acquisition method might be stored remotely.**

## ICF and RC.NET Instrument Control

Agilent developed RapidControl .NET (RC.NET) and Instrument Control Framework (ICF), which are two strongly related standards developed to cover the requirements of fully integrated instrument control.

RC.NET defines a standard for the implementation of hardware drivers on a **module** level, whereas ICF primarily defines the interfacing for the integration of **instrument** control into an application, on an instrument level. Instrument level means that the ICF aggregates multiple RC.NET drivers to a combined instrument view.

### Overall Architecture

The general abstraction layers, illustrated in previous figures, evolve to the following layers, if the RC.NET and ICF concepts are included.

The application and the instrument layers remain the same, whereas the instrument driver and the application adapter can be modeled with finer granularity as required (Figure 4). The instrument driver layer is substituted by the RC.NET drivers to control the modules, independent of the host CDS or workstation architecture. For example, instrument drivers are directly supplied by Agilent and are specific to the respective modules. The application adapter is required to integrate the functions of ICF transparently into the host CDS or workstation. The complexity of the adapter is minor, as it only integrates ICF, which provides the integrated functionality for full instrument control.

The benefits of this architecture are as follows.

- The RC.NET standard allows the implementation of driver support for hardware modules fully independent from an application.

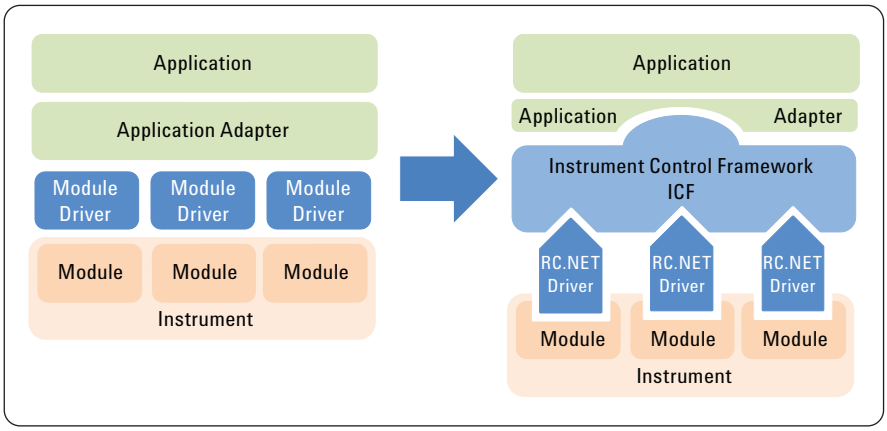



**Figure 4**
**Instrument control abstraction layers using ICF and RC.NET.**

- The ICF builds up layers on top of RC.NET drivers, which provides an aggregation of multiple RC.NET drivers resulting in an instrument view. Due to the aggregation of multiple drivers the application adapter layer is more lightweight, since parts of the functionality and synchronization tasks are already covered within the ICF layer.

### ICF and RC.NET Architectural principles

The fundamental RC.NET and ICF paradigms are designed to allow the use of the ICF components in a wide application area, independent from deployment and application infrastructure. They are used in the following ways:

- The components can be used as independent building blocks in the hosting application.
  - The complete communication between different components is based on XML exchange between the components, using the underlying application infrastructure.
  - The representation components (UI components) are separated from functional components.

    This provides the maximum flexibility for the different usage scenarios of the hosting application
- The relevant data containers (configuration, method, pretreatment, status) are encapsulated and handled as black boxes.
  - Data containers are modeled as XML fragments to support the communication paradigms of both standards, but the structure and content of these XML fragments must not be used by the host application.
  - The host application is de-coupled from driver specific settings. This removes the need of knowing / using driver specific commands on a programmatic level, including dependencies from the driver versions.
  - Data migration interfaces ensure that the contents of the data containers can be migrated from older versions to newer in a generic way.
  - Data access interfaces to the data containers provide the generic data access to the containers using reflection-like mechanism if necessary. The advantage is that the data structures can be accessed dynamically; for example, queried and modified as required.

- The run control is decoupled from the sequence execution and data analysis functionality.
    - Scope of the run control is a single sample analysis. Further automation aspects (such as sequence control and execution) must be covered by the application, resulting in the possibility of component use in different automation scenarios.
    - Data from sample acquisitions are returned transparently to the host application as, for example, unprocessed format, delivered by the instrument. Further data manipulation and presentation steps are completely separated from the interfaces and are located in the scope of the application.

In addition to these common paradigms the ICF introduces further paradigms:

- ICF follows an instrument centric paradigm. This means the components and the data handled by the components always have the granularity of an instrument contrary to a single module view. Instrument view in the context of IFC means that any module within a system that is controlled by an RC.NET driver, is part of the instrument, published by the ICF.
- ICF provides a plug-in architecture for RC.NET drivers:
    - No additional implementation effort is necessary to support new modules. A new module is automatically available for the application after installation.
    - Easy deployment. The customer only has to install the new RC.NET driver and he is then able to use the new module within his application.

## Conclusion

ICF and RC.NET are complementary instrument control interface components.

- ICF is completely built upon the RC.NET drivers and is therefore dependent on the availability of the RC.NET driver to support a distinct hardware.

  ICF introduces a layer, abstracting from the single RC.NET drivers and provides a combined view of all participating RC.NET drivers, linked together to represent a complete instrument with full synchronization.
- In contrast, RC.NET is designed to control a single module. RC.NET provides the foundation for implementing drivers that in turn can be used in a generic way. This allows the development of drivers to support modules independent from the application.

Exploiting the strength of both paradigms leads to the following domain characterizations:

- RC.NET provides a technique for implementing distinct hardware support independent of the application.
- ICF provides a technique for integrating a generic support for instruments in a distinct application.

The concept of ICF is to be connected once to a non-Agilent chromatographic data system or workstation as a piece of middle ware, which allows subsequent transparent connection of Agilent LC instrument drivers to the third party data system. From an instrument control perspective, ICF provides functionality that is identical to that of ChemStation because Agilent native instrument drivers are used. However, they are fully integrated in third party software. ICF is not the tool to change or extend capabilities on a module level, because the design of ICF integrates existing drivers in third party CDS or workstations. It is possible to **extend/modify** instrument control capabilities, via tools that have been available for some time (for example, LICOP, ChemStation macro programming).

## References

"The Agilent Instrument Control Framework (ICF): A short introduction for end users of Agilent HPLC instrumentation as well as third-party instrument control providers." Agilent Technologies Data Sheet, publication 5990-5756EN.

# Appendix

## ICF abstraction layers

The different levels of communication protocols between the layers show increasing simplification of the protocol with a modular instrument (Figure 5).

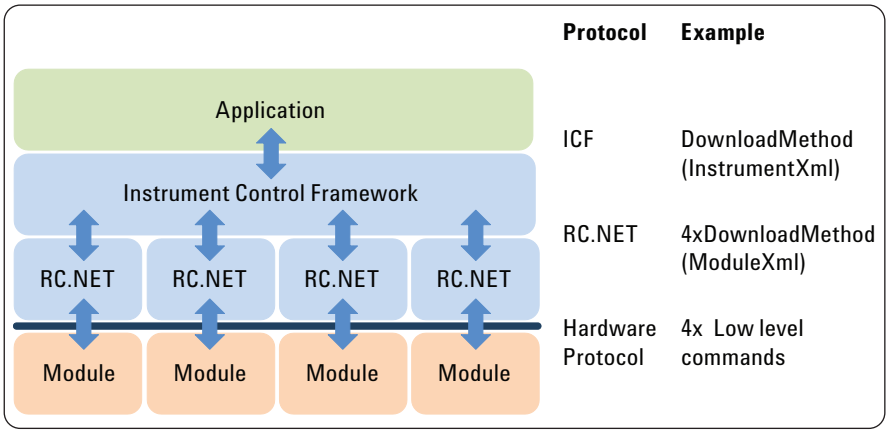



**Figure 5**
**ICF abstraction layers.**

## ICF functional blocks

The ICF provides important components that cover typical aspects of the data system mentioned above (Figure 8):

- Method component:

  Method handling functionality (audit trail creation for two methods, reporting of method)

- Method UI component:

  UI for editing method parameters (method parameter, timetable) including help and tool tips

- Control component:

  Interaction with the hardware and run control functionality (downloading method, start/stop run, gathering chromatography data)

- Configuration component:

  Configuration handling functionality (audit trail creation for two configurations, reporting of configuration)

- Configuration UI component:

  UI for editing the configuration including help and tool tips

- Status UI component (optional):

  UI to show the overall status of the hardware including direct interaction for dedicated features (switch on/off UV lamp)

- Pretreatment component (optional):

  Pretreatment handling parameters, if supported by the hardware. Pretreatment defines customized sample handling steps for samplers. In some applications this feature is also known as the injector program.

- Pretreatment UI component (optional):

  UI for editing the pretreatment parameters

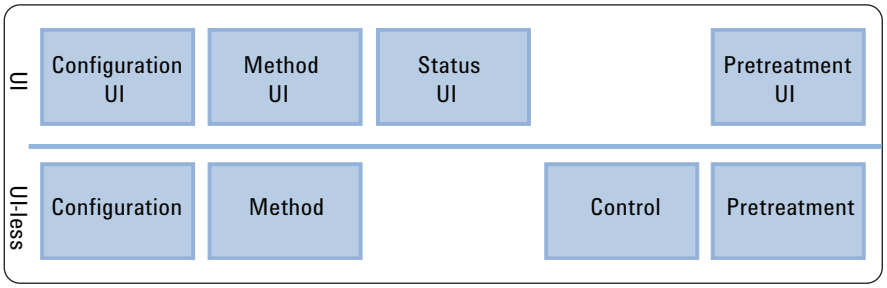



**Figure 6**
**ICF functional blocks.**

In addition to the components above, ICF provides support for extended use cases, such as actions, or well plate editing.

## ICF deployment models

ICF defines the components as independent building blocks that can be used and wired by an application with the application-dependent infrastructure and deployment strategy (for example, workstation, or client/server environment). The complete communication between the components occurs by passing XML fragments between the different components. There is no other direct linkage between these components such as DCOM, or RPC. This makes these components compatible for different deployment strategies.

The communication in a deployment scenario using a separate acquisition controller is illustrated in Figure 7.

## ICF driver packages

The deployment strategy for the ICF is based upon two parts:

- Framework Installation

  This installs a version of the ICF components within the context of the host application. Installing the framework allows an application to use ICF driver packages.

  The framework is designed to be installed completely without user interaction. Therefore, it can be installed in the context of the host application's install process.

- Driver Package Installation

  This installs a number of driver packages. A driver package defines the available modules that can be used if that package has been chosen during configuration of the instrument.

During startup of the framework, the framework scans the system for installed packages, which are available for use within ICF.

## ICF cooperation with non RC.NET modules

Although ICF provides the system view for RC.NET capable modules, it is still possible to run "mixed" instruments, containing modules without RC.NET drivers. In this case the host application must synchronize the ICF instrument with the participating non RC.NET modules. In this scenario you can think of the ICF layer as a "complex" module driver (Figure 8).

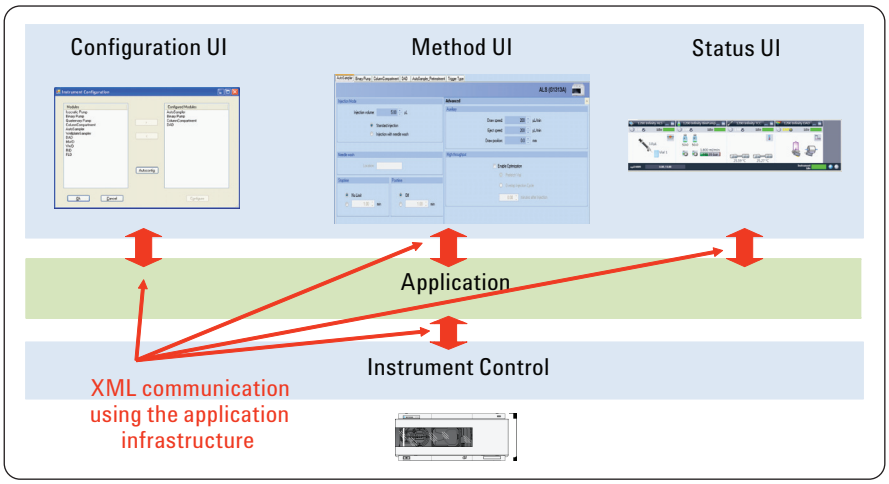



**Figure 7**
**Communication using separate acquisition controller.**

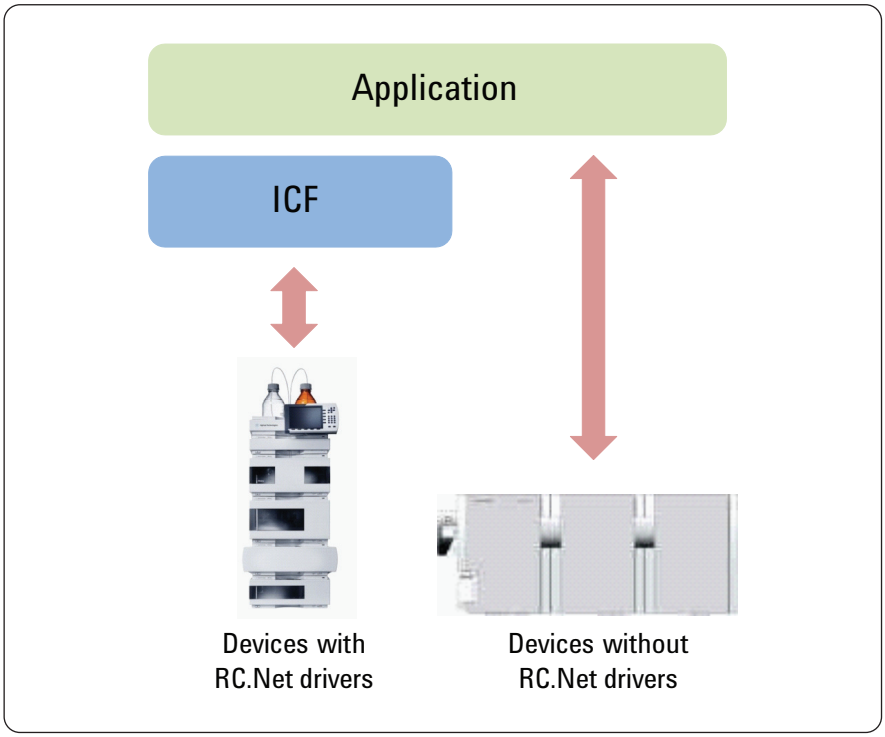



**Figure 8**
**Schematic system view of "mixed" instrument enviroment.**

www.agilent.com/chem/lc

© Agilent Technologies, Inc., 2011
February 1, 2011
Publication Number 5990-6504EN

Agilent Technologies